IN THE UNITED STATES DISTRICT COURT
FOR THE MIDDLE DISTRICT OF FLORIDA
ORLANDO DIVISION

TESSERON, LTD.,

Plaintiff,

v.

PUNCH INTERNATIONAL NV, PUNCH GRAPHIX NV, PUNCH GRAPHIX AMERICAS INC., GLOBAL GRAPHICS SA, AND GLOBAL GRAPHICS SOFTWARE, INC.

Defendants.

CASE NO.: 6:10-cv-909-Orl-31GJK

COMPLAINT FOR PATENT INFRINGEMENT

DEMAND FOR JURY TRIAL

Plaintiff Tesseron, Ltd. ("Tesseron"), for its complaint against Defendants Punch International NV ("Punch"), Punch Graphix NV ("Punch Graphix"), Punch Graphix Americas Inc. ("Punch Graphix Americas"), Global Graphics SA ("Global Graphics"), and Global Graphics Software, Inc. ("Global Graphics Software"), alleges as follows:

JURISDICTION AND VENUE

1. This is an action for patent infringement arising under the laws of the United States, 35 U.S.C. § 1, et seq.

2. This Court has subject matter jurisdiction of this action based on 28 U.S.C. § 1331, which provides: "[t]he district courts shall have original jurisdiction of all civil actions arising under the Constitution, laws, or treaties of the United States." This Court additionally has subject matter jurisdiction of this action under 28 U.S.C. § 1338(a), which

provides: "[t]he district courts shall have original jurisdiction of any civil action arising under any Act of Congress relating to patents, plant variety protection, copyrights and trademarks."

3. This Court has general personal jurisdiction over each of the defendants under Florida's long-arm statute, Fla Stat. § 48.193, because each defendant maintains continuous and systematic contacts with Florida and engages in substantial and not isolated activity within Florida. Further, this Court may exercise specific personal jurisdiction over each defendant under Fla Stat. § 48.193 because the distribution and sale of infringing products, as described below, constitute:

a. operating, conducting, engaging in, or carrying on a business or business venture in Florida and/or having an office or agency in this state; and

b. committing a tortious act within Florida.

4. Under Chapter 87 of Title 28 of the U.S. Code, "a defendant that is a corporation shall be deemed to reside in any judicial district in which it is subject to personal jurisdiction at the time the action is commenced." 28 U.S.C. § 1391(c).

5. Consequently, venue is proper in this district under 28 U.S.C. § 1400(b), which provides: "[a]ny civil action for patent infringement may be brought in the judicial district where the defendant resides, or where the defendant has committed acts of infringement and has a regular and established place of business."

PARTIES

A. Tesseron

6. Plaintiff Tesseron is an Ohio limited liability company with its principal place of business in Orlando, Florida.

B. Punch, Punch Graphix, and Punch Graphix Americas

7. Defendant Punch Graphix is a corporation of the Netherlands with its principal place of business at Brieversstraat 70, 4529 GZ Eede, the Netherlands. Punch Graphix is selling, marketing, and/or importing variable-enabled printing systems in this judicial district and elsewhere throughout the United States.

8. Defendant Punch is incorporated under the laws of Belgium with its principal place of business in Belgium. Upon information and belief, Punch acquired Xeikon International N.V. ("Xeikon") in 2002,[1] owns or controls Punch Graphix, and—either directly or through entities that Punch owns and/or controls—manufactures and/or distributes variable-enabled printing systems and has introduced infringing products into the stream of commerce knowing they would be sold in this jurisdiction and elsewhere throughout the United States.

9. Defendant Punch Graphix Americas is a Delaware corporation with its principal place of business in Illinois. Punch Graphix Americas is selling, marketing, and/or importing variable-enabled printing systems in this judicial district and elsewhere throughout the United States.

[1] Xeikon International N.V. was a Belgian corporation that sold, marketed, and/or imported variable-enabled printing systems in this judicial district and elsewhere throughout the United States.

10. Upon information and belief, Punch Graphix owns or controls Punch Graphix Americas, and Punch Graphix Americas is acting as the agent of Punch and Punch Graphix for purposes of conducting the business of Punch and Punch Graphix in this jurisdiction and elsewhere throughout the United States. Upon information and belief, Punch Graphix Americas—either directly or through entities that Punch Graphix Americas owns and/or controls—also manufactures and/or distributes variable-enabled printing systems and has introduced infringing products into the stream of commerce knowing they would be sold in this jurisdiction and elsewhere throughout the United States.

C. Global Graphics and Global Graphics Software

11. Defendant Global Graphics Software is a Massachusetts corporation with its principal place of business in Massachusetts. Global Graphics Software is registered to do business as a foreign corporation in Florida and is selling, marketing, and/or importing variable-enabled printing systems in this judicial district and elsewhere throughout the United States.

12. Defendant Global Graphics is a corporation of France with its principal place of business in Pompey, France. Upon information and belief, Global Graphics owns or controls Global Graphics Software, and Global Graphics Software is acting as the agent of Global Graphics for purposes of conducting the business of Global Graphics in this jurisdiction and elsewhere throughout the United States. Upon information and belief, Global Graphics—either directly or through entities that Global Graphics owns and/or controls—also manufactures and/or distributes variable-enabled printing systems and has

introduced infringing products into the stream of commerce knowing they would be sold in this jurisdiction and elsewhere throughout the United States.

FIRST CLAIM FOR RELIEF
(Infringement of U.S. Patent No. 6,381,028 B1)

13. Tesseron incorporates by reference Paragraphs 1 through 12 as though fully alleged and set forth herein.

14. On April 30, 2002, United States Patent No. 6,381,028 B1 ("the '028 patent") titled "Method of Utilizing Variable Data Fields with a Page Description Language" was duly and legally issued to Tesseron. A copy of the '028 patent is attached hereto as Exhibit 1.

15. On May 24, 2001, the '028 patent (then pending application U.S. Ser. No. 09/299,502) was assigned from Varis Corporation to Tesseron. The assignment was duly recorded in the U.S. Patent and Trademark Office on July 2, 2001, at reel/frame 011944/0233.

A. Punch, Punch Graphix, and Punch Graphix Americas

16. Punch, Punch Graphix, and Punch Graphix Americas (the "Punch Defendants") have infringed and continue to infringe, have actively induced and currently are actively inducing others to infringe, and/or have contributorily infringed and currently are contributorily infringing at least one claim of the '028 patent in the United States by making, using, selling, offering to sell and/or importing for sale into the United States products that embody the inventions described and claimed in the '028 patent, including, but not limited to, Xeikon X-800 Digital Front End alone or in combination with other Xeikon components

such as Xeikon 330, Xeikon 3300/3000, Xeikon 4000, Xeikon 5000, Xeikon 5000Plus, Xeikon 6000, Xeikon 8000, Xeikon IntelliStream 3.0, 3.5, 3.6, and all other versions.

17. Upon information and belief, the Punch Defendants' infringement of claim(s) of the '028 patent has been willful, deliberate, and in conscious disregard of Tesseron's rights, making this an exceptional case within the meaning of 35 U.S.C. § 285.

18. As a result of the actions of the Punch Defendants, Tesseron has suffered and will continue to suffer substantial injury, including irreparable harm and damages, and loss of sales and profits that Tesseron would have made but for the infringement by the Punch Defendants, unless the Punch Defendants are preliminarily and/or permanently enjoined by this Court.

B. Global Graphics and Global Graphics Software

19. Upon information and belief, Global Graphics and Global Graphics Software (the "Global Graphics Defendants") have infringed and continue to infringe, have actively induced and currently are actively inducing others to infringe, and/or have contributorily infringed and currently are contributorily infringing at least one claim of the '028 patent in the United States by making, using, selling, offering to sell and/or importing for sale into the United States products that embody the inventions described and claimed in the '028 patent, including, but not limited to, the Harlequin Raster Image Processor.

20. Upon information and belief, the Global Graphics Defendants' infringement of claim(s) of the '028 patent has been willful, deliberate, and in conscious disregard of Tesseron's rights, making this an exceptional case within the meaning of 35 U.S.C. § 285.

21. As a result of the actions of the Global Graphics Defendants, Tesseron has suffered and will continue to suffer substantial injury, including irreparable harm and damages, and loss of sales and profits that Tesseron would have made but for the infringement by the Global Graphics Defendants, unless the Global Graphics Defendants are preliminarily and/or permanently enjoined by this Court.

SECOND CLAIM FOR RELIEF
(Infringement of U.S. Patent No. 6,771,387 B2)

22. Tesseron incorporates by reference Paragraphs 1 through 15 as though fully alleged and set forth herein.

23. On August 3, 2004, United States Patent No. 6,771,387 B2 ("the '387 patent") titled "Method of Utilizing Variable Data Fields with a Page Description Language" was duly and legally issued to Tesseron. A copy of the '387 patent is attached hereto as Exhibit 2.

24. On May 24, 2001, the '387 patent (as a yet un-filed continuation of U.S. Patent 5,937,153 and the '028 patent) was assigned from Varis Corporation to Tesseron. The assignment was duly recorded in the U.S. Patent and Trademark Office on July 2, 2001, at reel/frame 011944/0233.

25. The U.S. patent application for the '387 patent was published by the U.S. Patent and Trademark Office on September 5, 2002, as US 2002/122205 A1.

A. The Punch Defendants

26. Upon information and belief, the Punch Defendants have infringed and continue to infringe, have actively induced and currently are actively inducing others to infringe, and/or have contributorily infringed and currently are contributorily infringing at

least one claim of the '387 patent in the United States by making, using, selling, offering to sell and/or importing for sale into the United States products that embody the inventions described and claimed in the '387 patent, including, but not limited to, Xeikon X-800 Digital Front End alone or in combination with other Xeikon components such as Xeikon 330, Xeikon 3300/3000, Xeikon 4000, Xeikon 5000, Xeikon 5000Plus, Xeikon 6000, Xeikon 8000, Xeikon IntelliStream 3.0, 3.5, 3.6, and all other versions.

27. Upon information and belief, the Punch Defendants' infringement of claim(s) of the '387 patent has been willful, deliberate, and in conscious disregard of Tesseron's rights, making this an exceptional case within the meaning of 35 U.S.C. § 285.

28. As a result of the actions of the Punch Defendants, Tesseron has suffered and will continue to suffer substantial injury, including irreparable harm and damages, and loss of sales and profits that Tesseron would have made but for the infringement by the Punch Defendants, unless the Punch Defendants are preliminarily and/or permanently enjoined by this Court.

B. The Global Graphics Defendants

29. Upon information and belief, the Global Graphics Defendants have infringed and continue to infringe, have actively induced and currently are actively inducing others to infringe, and/or have contributorily infringed and currently are contributorily infringing at least one claim of the '387 patent in the United States by making, using, selling, offering to sell and/or importing for sale into the United States products that embody the inventions described and claimed in the '387 patent, including, but not limited to, the Harlequin Raster Image Processor.

30. Upon information and belief, the Global Graphics Defendants' infringement of claim(s) of the '387 patent has been willful, deliberate, and in conscious disregard of Tesseron's rights, making this an exceptional case within the meaning of 35 U.S.C. § 285.

31. As a result of the actions of the Global Graphics Defendants, Tesseron has suffered and will continue to suffer substantial injury, including irreparable harm and damages, and loss of sales and profits that Tesseron would have made but for the infringement by the Global Graphics Defendants, unless the Global Graphics Defendants are preliminarily and/or permanently enjoined by this Court.

PRAYER FOR RELIEF

WHEREFORE, Tesseron prays that the Court enter judgment in its favor as follows:

1. That Defendants be declared and adjudged to have infringed, actively induced others to infringe, and/or contributorily infringed one or more claims of U.S. Patent Nos. 6,381,028 and 6,771,387;

2. That Defendants, their agents, sales representatives, distributors, servants and employees, attorneys, affiliates, subsidiaries, successors and assigns, and any and all persons or entities acting at, through, under or in active concert or in participation with any or all of them, be enjoined and restrained preliminarily and permanently, from infringing, actively inducing others to infringe, and/or contributorily infringing claim(s) of U.S. Patent Nos. 6,381,028 and 6,771,387;

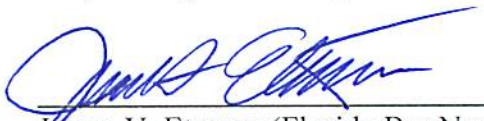
3. An accounting be had for the profits and other damages arising out of Defendants' infringement of U.S. Patent Nos. 6,381,028 and 6,771,387, and that the damages

be trebled pursuant to 35 U.S.C. § 284 for the willful acts of infringement complained of herein and awarded to Tesseron, together with prejudgment and post-judgment interest;

4. That this case be decreed an exceptional case within the meaning of 35 U.S.C. § 285 and reasonable attorneys' fees be awarded to Tesseron; and

5. That Tesseron be awarded such other costs and further relief as the Court deems just and proper.

Respectfully submitted,



James V. Etscorn (Florida Bar No: 705111)
Email: jetscorn@bakerlaw.com
BAKER & HOSTETLER LLP
2300 SunTrust Center
200 South Orange Avenue
Orlando, FL 32801-3432
Telephone: 407-649-4000

Kevin W. Kirsch (*pro hac vice* to be filed)
Email: kkirsch@bakerlaw.com
David A. Mancino (*pro hac vice* to be filed)
Email: dmancino@bakerlaw.com
John F. Bennett (*pro hac vice* to be filed)
Email: jbennett@bakerlaw.com
Matthew P. Hayden (*pro hac vice* to be filed)
Email: mhayden@bakerlaw.com
BAKER & HOSTETLER LLP
312 Walnut St., Suite 3200
Cincinnati, OH 45202-4074
Telephone: (513) 929-3400

ATTORNEYS FOR PLAINTIFF
TESSERON, LTD.

JURY DEMAND

Plaintiff Tesseron demands trial by jury on all issues.

James V. Etscorn (Florida Bar No: 705111)

097708, 000002, 502903030.2

EXHIBIT 1

United States Patent No. 6,381,028 B1

(12) **United States Patent**
Gauthier

(10) **Patent No.: US 6,381,028 B1**
(45) **Date of Patent: *Apr. 30, 2002**

(54) **METHOD OF UTILIZING VARIABLE DATA FIELDS WITH A PAGE DESCRIPTION LANGUAGE**

(75) Inventor: **Forrest P. Gauthier**, Maineville, OH (US)

(73) Assignee: **Tesseron Ltd.**, Loveland, OH (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **09/299,502**

(22) Filed: **Apr. 26, 1999**

Related U.S. Application Data

(63) Continuation of application No. 08/896,899, filed on Jul. 18, 1997, now Pat. No. 5,937,153, which is a continuation-in-part of application No. 08/373,582, filed on Jan. 18, 1995, now Pat. No. 5,729,665.

(51) **Int. Cl.**[7] **G06K 15/00**
(52) **U.S. Cl.** **358/1.11**; 358/1.18
(58) **Field of Search** 358/1.1, 1.9, 1.15, 358/1.18, 1.19, 1.7, 1.11, 1.13, 501, 522, 702, 706, 709, 448, 467; 382/318, 164, 191

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,103,490	A	4/1992	McMillin	382/62
5,134,669	A	7/1992	Keogh et al.	382/61
5,157,765	A	10/1992	Birk et al.	395/163
5,202,206	A	4/1993	Tam	430/41
5,222,236	A	6/1993	Potash et al.	395/600
5,291,243	A	3/1994	Heckman et al.	355/201
5,459,819	A	10/1995	Watkins et al.	395/117
5,459,826	A	10/1995	Archibald	395/147
5,506,697	A	4/1996	Li et al.	358/448
5,729,665	A *	3/1998	Gauthier	358/1.1
5,740,338	A *	4/1998	Gauthier et al.	358/1.17
5,852,673	A *	12/1998	Young	382/164
5,937,153	A *	8/1999	Gauthier	358/1.18
6,016,380	A *	1/2000	Norton	358/335

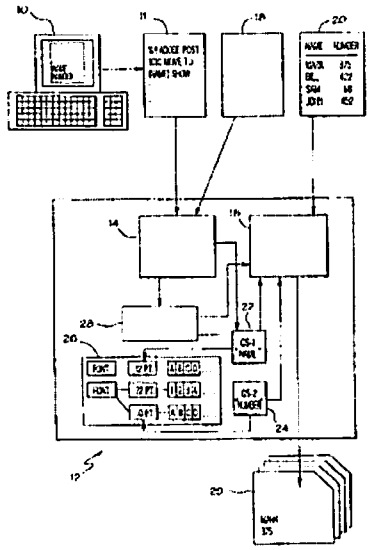
* cited by examiner

Primary Examiner—Gabriel Garcia
(74) *Attorney, Agent, or Firm*—Taft, Stettinius & Hollister LLP

(57) **ABSTRACT**

A computer implemented method for generating a plurality of bit maps suitable for high-speed printing includes the steps of: (a) providing a page description code specification, where the page description code specification defines at least one data area, and the page description code further defines a graphics state corresponding to the data area, where the graphics state including at least one attribute which controls the appearance of data in the data area; (b) interpreting the page description code specification, and during the interpretation step, identifying the data area defined by the page description code specification; (c) upon the identification of the variable data area in step (b), applying the graphics state corresponding to the data area to a set of alphanumeric characters so as to generate a plurality of character bit maps; (d) storing the plurality of character bit maps; (e) retrieving a variable data item from a plurality of variable data items; (f) associating the variable data item with the plurality of character bit maps; (g) generating a variable data bit map for the variable data using the character bit maps; and (h) repeating steps (e) through (g) for remaining variable data items in the plurality of variable data items. Thus, the stored character bit maps are used repeatedly to generate a plurality of variable data bit maps.

4 Claims, 2 Drawing Sheets



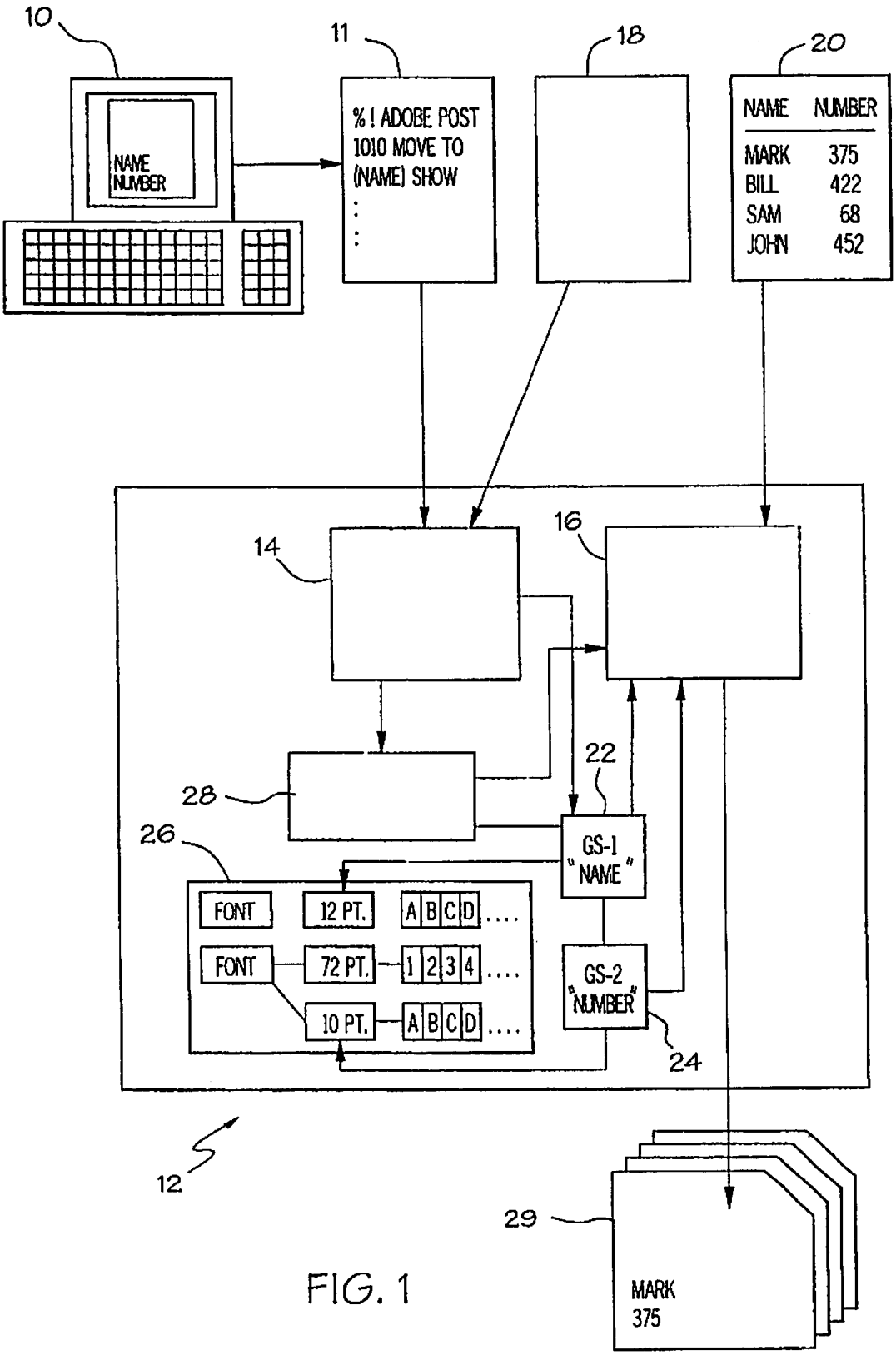

10
NAME
NUMBER
11
%! ADOBE POST
1010 MOVE TO
(NAME) SHOW
18
20
NAME NUMBER
MARK 375
BILL 422
SAM 68
JOHN 452
14
16
22
28
26
GS-1
"NAME"
FONT
12 PT.
A B C D ....
FONT
72 PT.
1 2 3 4 ....
10 PT.
A B C D ....
GS-2
"NUMBER"
24
12
29
MARK
375

FIG. 1

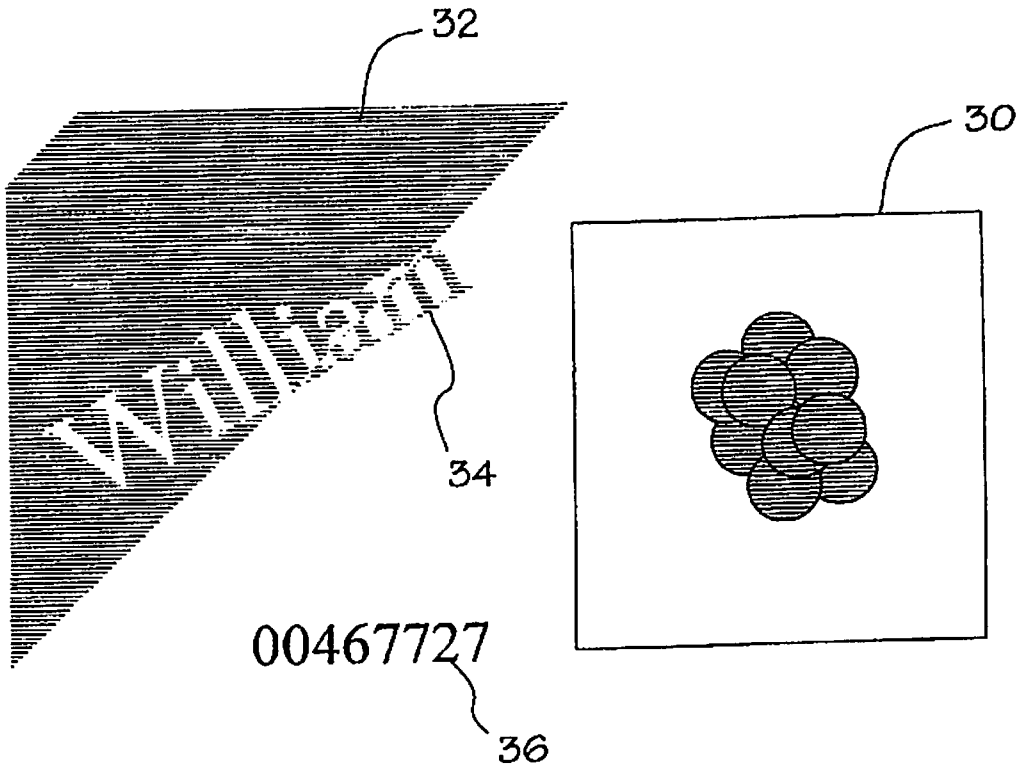


FIG. 2

METHOD OF UTILIZING VARIABLE DATA FIELDS WITH A PAGE DESCRIPTION LANGUAGE

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is a continuation of U.S. application, Ser. No. 08/896,899, filed Jul. 18, 1997, and issued as U.S. Pat. No. 5,937,153; which is a continuation-in-part of U.S. application Ser. No. 08/373,582, filed Jan. 18, 1995, and issued as U.S. Pat. No. 5,729,665.

BACKGROUND OF THE INVENTION

The present invention relates to the high-speed printing industry, and more particularly, to a method for printing variable data using a page description language in a high-speed printing environment.

Application programs, such as Adobe Illustrator®, typically include a program which generates a specification of a screen or page's contents in a page description language. The specification, or page description code, provides instructions as to how to generate the image in a printer. The page description code is transferred from the application program to a printer, where it is executed to generate a bit map of the page. The most commonly used page description language is PostScript®, which is a machine independent language produced by Adobe Systems, Inc.

An application program page typically contains a number of data areas with either graphic or alphanumeric data. The PostScript language includes commands that define or build "graphics states" for each of the data areas on the page. These graphics states are sets of default attributes such as angle, scale factor, type-font, location, etc., which define how data is to appear on the page. Often, multiple graphics states are defined for a single page, with the different graphic states corresponding to different data areas on the page. Examples of commands that are used in PostScript to build a graphics state are: 20 rotate, /Times-Roman findfont, 14 scalefont, and setfont. In addition to commands which build graphics states, PostScript specifications also include the graphic or alphanumeric data which is displayed in the data areas, as well as a print command such as "SHOW", which causes a bit map to be generated for the data.

In the past, page description languages, including PostScript, have only been used to print static data pages, because page description languages lack the functionality required for variable data printing. In variable data printing, each page shares a common background, and the displayed data in at least one data field changes for each page. Up until now, it has not been possible to print pages of variable data with page description languages such as PostScript, because the page description languages are unable to save page backgrounds and graphics states from a page specification, and are thus unable reuse the same background and graphics states when printing subsequent pages. Thus, with page description languages such as PostScript, whether the entire page is changed, or only a single item of data on the page is changed, a new page description language specification is generated to print each separate page.

For example, if thousands of copies of a mass mailing advertisement were to be printed, each copy being identical except for the recipient's name and address, it would be necessary to generate a new PostScript specification defining the page background, and the graphics states for the name and address fields, for each new name and address that is printed. Hence, to print 50 advertisements, it would be necessary to generate 50 PostScript specifications which each define virtually the same image.

In general, PostScript specifications are very complex and require extensive processing to generate and execute. Thus, generating a new PostScript specification each time a page of variable data is printed consumes an immense amount of processing time. In high-speed printing systems, it is typically the processing time, not the printer hardware, which determines the speed at which pages can be printed. Therefore, the processing required to repetitively redefine the same background and graphics states for each page of variable data significantly slows the entire printing system.

Due to the amount of processing time consumed in redefining the page template and graphics states for each new page of data that is printed, as well as the resultant effect on printing speed, it is desirable to have a method for processing variable data wherein once defined, the template and graphics states for a page can be stored and reused for printing subsequent pages. Further, it is desirable to have a method for printing variable data which is compatible with existing printing systems and page description languages, such as PostScript, and which is capable of processing variable data in a high-speed industrial printing system.

SUMMARY OF THE INVENTION

It is an object of the present invention to provide a method for utilizing variable data with a page description language, which enables the template and graphics states for a page of variable data to be defined and stored; and which enables the stored graphics states to be associated with multiple items of variable data from a database or merge file; so that once stored, the graphics states can be repeatedly applied to the items of variable data to print multiple pages of variable data or multiple variable data bitmaps. Further, it is an object of the present invention to provide such a method which is compatible with existing page description languages, and which can be used in a high-speed industrial printing system.

The method of the present invention is implemented by means of a control task which executes in conjunction with a page description code interpretive program, such as a PostScript program, to identify variable data areas in the page description code specification, and reserve the graphics states for the variable data areas as they are defined by the specification. After the interpreter program has executed, a merge task is initiated. The merge task associates items of variable data from a data file with the reserved graphics states, generates a bit map for each variable data area, merges the bit maps with the page template, and outputs a complete bit map for the page. Accordingly, in the method of the present invention, bit maps for multiple pages of variable data are generated from a single page description language specification.

The present invention assumes the generation of a page specification in PostScript, or another similar page description language, by the application program, and the transfer of this specification to a printer. According to the present invention, a control task activates and monitors the PostScript interpreter program in the printer. As the interpreter executes, it defines graphics states for the data areas on the page. The PostScript attributes for a graphics state are stored in a stack as they are defined, so that at any given point in the code, the stack represents all of the PostScript attributes for the current graphics state.

When the control task identifies a print command in the code, the control task interrupts the interpreter to determine whether the data to be printed is variable data. If the data is

variable, the current graphics state, consisting of the attributes then existing in the stack and job specific attributes which are defined in a job file, is linked to the data area and reserved in an internal database. Further, character bit maps are generated in accordance with the graphics state, and linked to and reserved with the graphics state. After the graphics state and character bit maps have been reserved, the PostScript interpreter is resumed at the line of code following the print command.

The interpreter continues executing until either the control task detects another print command, or the last line of code is reached. If a second print command is detected, the interpreter is interrupted again and the above steps repeated, to reserve the stack contents and job attributes for the second data area, and to generate and store a second set of character bit maps. The control task continues in this manner monitoring and interrupting the interpreter program, until all of the variable data areas on the page have been detected, and graphics states and possibly character bit maps for the variable data areas have been reserved in the database.

As the PostScript interpreter executes, a bit map of the non-variable background graphics and text, otherwise referred to as a "template", is generated for the page. At the last code command, which in PostScript is typically "SHOWPAGE," the control task terminates the PostScript interpreter, and reserves the template in the database.

The merge task is then initiated to print variable data pages using the reserved page template, graphics states and character bit maps. The merge task begins by retrieving a merge file containing the variable data to be printed. After retrieving the merge file, the task identifies the correct template for the current page, and the names of the graphics states related to that template, from data in the merge file. Then, using the name of the first graphics state reserved for the template, the merge task retrieves the graphics state from the database and the character bit maps linked to that state. The merge task then retrieves data corresponding to that graphics state from the appropriate field in the merge file, and generates a bit map of the data in accordance with the graphics state and character bit maps. The merge task then merges the data bit map into the template. After the bit map has been generated and merged, the merge task identifies retrieves another graphics state for the template and repeats the process. If there are no more graphics states which correspond to variable data areas on the page, the merge task outputs the finished bit map for the page.

After the first page of data has been printed, the merge task retrieves a "clean" template from the database, and again identifies the graphics states for the page. The merge task then retrieves the next record of variable data from the database, and generates variable data bit maps for each of the fields in the record, in accordance with the reserved graphics states and character bit maps which correspond to each of the fields. The merge task continues in this manner, identifying variable data areas and generating bit maps for the variable data in the merge file, until a page has been printed for each variable data record in the file.

The method of the present invention is advantageous in that once the graphics states and template have been defined for a variable data page, they can be reused to print multiple pages of variable data with only a minimal amount of additional processing.

Accordingly, it is an object of the present invention to provide a method for printing variable data with a page description language; a method which increases the speed at which variable data pages can be printed; a method which enables the printing attributes for a page to be saved and used for printing multiple pages of data; and a method which is compatible with existing page description languages and printing systems.

Other objects and advantages of the present invention will be apparent from the following description, the accompanying drawings and the appended claims.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a data flow diagram illustrating the preferred embodiment of the method of the present invention;

FIG. **2** is an example of a variable data page generated in accordance with the method of the present invention.

DETAILED DESCRIPTION

The present invention provides a computer-implemented method for reserving graphics states, which enables a page description language to be used for variable data printing. In accordance with the present invention, graphics states describing variable data areas are generated by a page interpreter program and reserved in an internal database. The graphics states are later accessed from the database and used for printing variable data pages. The method of the present invention can be employed with a conventional page description language, such as PostScript, to enable variable data pages to be printed with a minimum amount of processing.

As shown in FIG. **1**, an image containing text and/or graphics data is created at a workstation **10**, using a graphics application program such as Adobe Illustrator®. As the image is created, the application program displays the image on the workstation screen. When the image is complete and ready to be printed as a page, the application program generates a specification of the image in PostScript in a conventional manner.

After the PostScript file **11** is generated, it is transferred from the workstation **10** to a printer generally designated as **12**. In the printer **12**, a PostScript interpreter **14** is executed to generate a pagemap of the image. In the method of the present invention, a control task operates in the printer **12** to initiate the PostScript interpreter program **14** and a merge task **16**. The control task is initiated upon power-on of the printer **12**, and controls and coordinates the variable data printing.

As the PostScript interpreter **14** executes, it defines the PostScript graphics state attributes for the page. These attributes can include the size, font, position, orientation, and location in which the graphic or text data is to appear on the page. The specifics of the attributes which are available in PostScript to define how data is to appear on a page would be well-known to one skilled in the art. Therefore, further description of these PostScript attributes is not provided.

However, one of the PostScript attributes, namely the region, has been expanded in the present invention to allow for variable data printing. In the method of the present invention, the region attribute is used to define the boundaries or extent to which a variable data graphics state will be allowed to effect a page. The graphics state extent is an invisible boundary which acts as a clippath for the page, to prevent variable data for a particular graphics state from extending outside the intended boundaries of the graphics state. The region of the graphics state extent is defined without altering PostScript, by using an ordinary shape, which is created as part of the image, to define the region. In the present invention, the artist creates a shape represent-

ing the extent as part of the page image, and specifies a particular trigger color or gray scale for the shape. Thus, the artist could include a medium gray scale rectangle in the upper left-hand corner of the page, with the boundaries of the rectangle representing the extent which the artist has defined for the graphics state positioned at that comer of the page. The medium gray scale will then represent the trigger color, and will be specified as the trigger for the region attribute in a job file **18** in the printer **12**.

In addition, a second parameter in the job file **18** can be used to specify whether the rectangle should appear on the page, or whether it is being used only to define a graphics state extent. Thus, if the artist also wants the medium gray scale rectangle to appear on the printed page, this parameter enables the color to act as a trigger, yet not inhibit the artist's design. When the rectangle is interpreted during the method of this invention, the control task will detect the trigger color and will save an "invisible" boundary represented by the rectangular as part of the graphics state.

As the PostScript attributes are defined, they are placed in a stack. When a new attribute is defined, it is added to the top of the stack. When an attribute is deleted, it is removed from the stack. The combination of all of the attributes located in the stack at any point during the execution of the PostScript interpreter **14** constitutes the "current" graphics state for the page.

When the interpreter reaches a print command, such as "SHOW" in PostScript, the command triggers the control task to interrupt the interpreter program. During this interruption, the control task interprets data in the PostScript file **11** and reserves a graphics state if the data is variable. Normally in a PostScript file, data which is to appear on the printed document is enclosed within parentheses. Thus, the control task identifies data in the file **11** by locating parentheses in the code.

After the control task identifies the data, it interprets the data to determine whether it is static data, which is to be part of the page template, or variable data. To interpret the data, the control task first reads the data located in the parentheses and compares the data with a list of literal data strings stored in the job file **18**. The job file **18** contains a list of data strings which are each associated with the name of a graphics state and its corresponding data field in a merge file **20**. In the preferred embodiment, the graphics state name is the same as the field name in the merge file **20**. The merge file **20** contains variable data arranged in records, with each record corresponding to a different page. Each record contains one or more data fields, which each correspond to separate variable data areas on the page. The list of data strings and associated graphics state names is entered in the job file **18** by the print operator prior to initiating the print job. If the data from the PostScript file **11** matches a data string in the job file **18**, the control task replaces the data from the file **11** with the graphics state name associated with the matching data string. In this manner, the control task transforms static data in the PostScript file into a variable data field, by substituting a graphics state field name for the static data in the file.

In a second embodiment, the graphics state name corresponding to the data area is defined directly within the PostScript file **11**, by making the name part of the image that is created in the application program. In this embodiment, the name is enclosed within brackets in the file, such as "<<>>", to enable the control task to identify the data as defining a graphics state rather than being an ordinary data string. Thus, to define the graphics state "ADDRESS" within the PostScript file **11**, the following would appear before a show command in the code: "(<<ADDRESS>>)". This second embodiment is advantageous in that it does not require the control task to compare the file data with a data list in the job file **18**; however, it does require coordinating the graphics state and field names between the merge file **20** and the application program.

If the control task determines that the data corresponds to a variable area, it reads the current contents of the graphics state stack to determine the attributes to be used for printing data in that area. In addition to the PostScript attributes specified in the stack, the graphics state can also include attributes which are specifically tailored to variable data printing. These additional attributes can either appear after the graphics state name inside a "show" command, if the graphics state is defined directly in the PostScript file, or can be specified in the job file **18** prior to execution of the print job. These additional attributes specify how the variable data is to be positioned within the graphics state. The following is a list of the variable data attributes which can be specified for a print job:

Name:

A label used to identify the data to which the graphics state applies. A single datum may be inserted into more than one graphics state so this attribute is not unique to a single state.

Glyphs:

A list of character glyphs, both attributes and images, which are available for use in the graphics state. (e.g. an alphabet of 72 point Times-Roman bold italic characters).

Static Data:

Data to be used in the event that variable data is not available.

Identification:

A number used to uniquely identify a graphics state.

Justification:

How to handle the text left to right-left border, right border, centered or justified.

Alignment:

How to place the text vertically in the graphics state. This could be top, bottom or centered.

Word Wrapping:

Selects a word wrapping algorithm.

Dynamic Registration:

Information on how to determine the registration from one page to the next.

Logic Mode:

The manner in which the bitmap merge takes place. This is one of seven binary combination techniques.

DP Procedure:

A procedure (or program) used to manipulate the variable data just before the graphics state is applied.

Data Selection:

Which portions of the variable data to use.

Underline:

Selects underlined text.

When the control task is triggered to reserve a graphics state, the above listed attributes, if specified, are combined with the PostScript attributes from the stack, and reserved as a single graphics state under the name obtained from the PostScript file **11** or the job file **18** such as shown at **22**.

After the control task has compiled the attributes for the current graphics state, it may instruct PostScript to generate a font cache **26** for the graphics state. The font cache **26**

consists of a character bit map for each of the alphanumeric characters A-Z and 0–9 generated in the font specified in the graphics state. After PostScript has generated all of the character bit maps, and placed the bit maps in the font cache **26**, the font cache is linked to the graphics state **22**, and reserved in the database. After the control task has reserved the current graphics state **22** and the font cache **26** in the database, it resumes execution of the PostScript interpreter **14** at the first line of code after the print or "SHOW" command, so that the print command is not executed.

After the interpreter is resumed, it continues defining graphics state attributes for the page, until the control task detects another print or "SHOW" command. Upon detecting another print command, the control task again interrupts execution of the interpreter, and determines whether the data in the PostScript file **11** corresponds to a variable data area. If the data corresponds to a variable data area, the control task again substitutes a graphics state name from the job file **18** for the data in the PostScript file **11**, and reads the graphics state attributes from the stack and job file. The control task also instructs PostScript to generate another font cache, if the attributes of the current graphics state differ from the attributes of previously reserved graphics states. The current graphics state and font cache are then linked, and reserved in the database under the second graphics state name from the job file **18**, such as shown at **24**. If the data does not correspond to a variable data area, the control task resumes execution of the interpreter at the print command, so that a bit map for the data can be generated and added to the template.

At the final line of code, the template is complete, and incorporates all of the static text and graphic data that is to appear on the printed document. At this point, the control task terminates the interpreter, and saves the template to the database such as shown at **28**. In PostScript, the control task is triggered to save the template by the "SHOWPAGE" command.

Since the control task of the invention operates externally of the PostScript interpreter, the method of the present invention enables bit maps and graphics states to be generated by the interpreter in a conventional manner. However, rather than printing a completed page map at the end of the interpreter program, the method of this invention reserves the page maps, character bit maps and graphics states generated by the interpreter, in order that they may be subsequently accessed and used to print multiple pages of variable data.

After the interpreter has been terminated, the control task initiates the merge task **16**. The merge task **16** interfaces between the merge file **20**, which has been pre-programmed with items of variable data, and the database in which the templates, font caches and graphics states defined by the interpreter have been saved, in order to combine the variable data with a template on a single page map. The merge task **16** begins by accessing the merge file **20** to retrieve the name of the template for the page, and then retrieving the specified template from the database. In addition, the merge task **16** retrieves the names of the data fields and reserved graphics states which are associated with the selected template from the merge file **20**.

Using the name corresponding to the first graphics state on the page, the merge task **16** accesses the merge file **20** and retrieves the data stored under that field name in the first data record. In the representative merge file **20** shown in FIG. **1**, the field names are NAME and NUMBER.

After the merge task **16** has read the data corresponding to the designated field name, it retrieves the graphics state which was reserved under the same name, as well as the character bit maps which are linked to that graphics state. The merge task **16** then generates a bit map of the data in accordance with the graphics state attributes. After the bit map is generated, it is merged into the template at the region corresponding to the graphics state, by writing the data bit map over the existing template bit map.

It will be apparent to those of ordinary skill in the art that it is within the scope of the invention to write the data bit map over a clean page as opposed to the template bitmap. For example, if the template contains no static bitmap data, then it would not be necessary to save an empty bitmap of the template in the database as described above. Thus, it is within the scope of the invention that the PostScript file **11** defines only variable data areas and does not define any static data areas. Such a PostScript file is illustrated in FIG. **1**.

After the data from the first field has been merged into the template, the merge task **16** reads the name corresponding to a second variable data area from the merge file **20**, if a second variable area exists on the page. The merge task **16** then retrieves the graphics state and linked font cache having the same name as the second variable area. Next, using this name, the merge task **16** again accesses the merge file **20**, and reads the data from the field of the same name. The merge task **16** then generates a bit map for the data in accordance with the graphics state and font cache, and again merges the data bit map into the template **28**.

The merge task **16** continues the steps of identifying variable data areas for the template, retrieving graphics states and character bit maps corresponding to the variable areas, accessing variable data from the merge file **20**, and generating bit maps for the variable data, until bit maps have been generated and merged for all of the variable data to be included on the page. When a bit map has been generated for each variable data area, and merged with the template **28**, the pagemap is output for printing as shown at **29**.

The merge task **16** then proceeds with printing a second page using the same template and graphics states, but a different variable data record in the merge file **20**. To print the second page, the merge task **16** retrieves a "clean" template from the database. Next, the merge task **16** again identifies the name of the first variable data area for that template and retrieves the graphics state of the same name. Then, the merge task **16** reads the data for that field from the second record of the merge file **20**, and generates a bit map of the data using the retrieved graphics state attributes and character bit maps. Once the bit map is generated, the merge task **16** merges the bit map into the template by writing the bit map over the template at the location defined by the graphics state.

The merge task **16** then continues processing in this manner until bit maps have been generated and merged into the template for all of the graphics states reserved for the page. After all of the bit maps for the second page have been merged into the template, the page is printed. The merge task **16** continues, repeating these steps for each record of data in the merge file **20**, until all of the variable data records have been printed on a page.

FIG. **2** shows a variable data page printed in accordance with the method of this invention. On this page, the data fields **30** and **32** are static fields which are part of the page template. The data field **34** containing the name "William" is a variable data field. Different names such as Mark or Sam, from the merge file **20**, are printed in this field on subsequent pages. The font, angle and color contrast in which "William" is displayed are all aspects of the graphics

state which were defined and stored during the steps of the present invention. Data field **36** which contains the number "00467727" is a second variable data area on the page. Again, the data displayed in this area varies on each page, depending upon the contents of the merge file **20**.

While the method described constitutes a preferred embodiment of the invention, it is to be understood that the present invention is not limited to this precise form, and that variations may be made without departing from the scope of the invention.

What is claimed is:

1. A computer implemented method for generating a plurality of bit maps suitable for high-speed printing comprising the steps of:

(a) providing a page description code specification, the page description code specification defining at least one data area, and the page description code further defining a graphics state corresponding to the data area, the graphics state including at least one attribute which controls the appearance of data in the data area;

(b) interpreting the page description code specification, and during the interpretation step, identifying the data area defined by the page description code specification;

(c) upon the identification of the data area in step (b), applying the graphics state corresponding to the data area to a set of alphanumeric characters so as to generate a plurality of character bit maps;

(d) storing the plurality of character bit maps;

(e) retrieving a variable data item from a plurality of variable data items;

(f) associating the variable data item with the plurality of character bit maps;

(g) generating a variable data bit map for the variable data using the character bit maps; and

(h) repeating steps (e) through (g) for remaining variable data items in the plurality of variable data items, whereby the stored character bit maps are used repeatedly to generate a plurality of variable data bit maps.

2. The computer implemented method of claim **1**, wherein the page description code specification represents a template and includes a static data area, and the computer implemented method further comprises the steps of:

executing portions of the page description code specification corresponding to the static data area to generate a template bit map; and

merging each of the plurality of the variable data bit maps into clean copies of the template bit map to create a plurality of merged bit maps.

3. The computer implemented method of claim **1**, wherein the identifying step includes the step of detecting predefined characters within a text string defined in the page description code specification.

4. A computer implemented method for generating a reusable template bit map suitable for high-speed variable printing, comprising the steps of:

generating a page description code specification, the page description code specification defining at least one variable data area and at least one static data area;

interpreting the page description code specification, and during the interpretation step,

generating a bitmap of the static data area and adding the bitmap of the static data area to a template bitmap;

identifying the variable data area, and

responsive to the identification of the variable data, not adding a bitmap of the variable data area to the template bitmap; and

saving the template bitmap, whereby copies of the template bitmap can be continuously accessed to create a plurality of variable data bitmaps.

* * * * *

EXHIBIT 2

United States Patent No. 6,771,387 B2

(12) **United States Patent**
Gauthier

(10) **Patent No.: US 6,771,387 B2**
(45) **Date of Patent: Aug. 3, 2004**

(54) **METHOD OF UTILIZING VARIABLE DATA FIELDS WITH A PAGE DESCRIPTION LANGUAGE**

(75) Inventor: **Forrest P. Gauthier**, Maineville, OH (US)

(73) Assignee: **Tesseron, Ltd.**, Loveland, OH (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 21 days.

(21) Appl. No.: **10/090,074**

(22) Filed: **Mar. 1, 2002**

(65) **Prior Publication Data**

US 2002/0122205 A1 Sep. 5, 2002

Related U.S. Application Data

(63) Continuation of application No. 09/299,502, filed on Apr. 26, 1999, which is a continuation of application No. 08/896,899, filed on Jul. 18, 1997, now Pat. No. 5,937,153, which is a continuation-in-part of application No. 08/373,582, filed on Jan. 18, 1995, now Pat. No. 5,729,665.

(51) **Int. Cl.**[7] **G06F 15/00**; G06K 15/00
(52) **U.S. Cl.** **358/1.15**; 358/1.18
(58) **Field of Search** 358/1.1, 1.9, 1.15, 358/1.13, 1.17, 1.18, 448, 467, 470; 382/164, 191, 318

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,103,490 A	4/1992	McMillin	382/62
5,134,669 A	7/1992	Keogh et al.	382/61
5,157,765 A	10/1992	Birk et al.	395/163
5,202,206 A	4/1993	Tam	430/41
5,208,906 A	5/1993	Morgan	395/148
5,222,236 A	6/1993	Potash et al.	395/600

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

CA	2207840	12/1997
CA	2215094	4/1998
EP	0 703 524 B1	8/1997

OTHER PUBLICATIONS

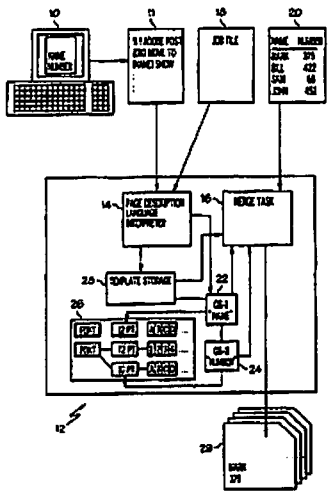
IDG InfoWorld, Oct. 9, 1995, Review, p. 91.
IDG InfoWorld, Mar. 20, 1995, Product Reviews; Desktop publishing software; p. 92.
InfoWorld, May 9, 1994, Reviews; p. 98.

Primary Examiner—Gabriel Garcia
(74) *Attorney, Agent, or Firm*—Taft, Stettinius & Hollister LLP

(57) **ABSTRACT**

A computer implemented method for generating a plurality of bit maps suitable for high-speed printing includes the steps of: (a) providing a page description code specification, where the page description code specification defines at least one data area, and the page description code further defines a graphics state corresponding to the data area, where the graphics state including at least one attribute which controls the appearance of data in the data area; (b) interpreting the page description code specification, and during the interpretation step, identifying the data area defined by the page description code specification; (c) upon the identification of the variable data area in step (b), applying the graphics state corresponding to the data area to a set of alphanumeric characters so as to generate a plurality of character bit maps; (d) storing the plurality of character bit maps; (e) retrieving a variable data item from a plurality of variable data items; (f) associating the variable data item with the plurality of character bit maps; (g) generating a variable data bit map for the variable data using the character bit maps; and (h) repeating steps (e) through (g) for remaining variable data items in the plurality of variable data items. Thus, the stored character bit maps are used repeatedly to generate a plurality of variable data bit maps.

42 Claims, 2 Drawing Sheets



U.S. PATENT DOCUMENTS

5,231,698 A	7/1993	Forcier	395/146
5,291,243 A	3/1994	Heckman et al.	355/201
5,459,819 A	10/1995	Watkins et al.	395/117
5,459,826 A	10/1995	Archibald	395/147
5,467,448 A	11/1995	Hilton et al.	395/148
5,483,624 A	1/1996	Christopher et al.	
5,500,928 A	3/1996	Cook et al.	
5,506,697 A	4/1996	Li et al.	358/448
5,539,529 A	7/1996	Merchant	358/400
5,542,052 A	7/1996	Deutsch et al.	395/131
5,544,287 A	8/1996	Roth	
5,600,768 A	2/1997	Andresen	395/135
5,611,024 A	3/1997	Campbell et al.	395/114
5,671,345 A	9/1997	Lhotak	395/133
5,754,750 A	5/1998	Butterfield et al.	395/118
5,841,420 A	11/1998	Kaply et al.	345/118
5,926,185 A	7/1999	Vyncke et al.	345/433
6,064,397 A	5/2000	Herregods et al.	
6,332,149 B1 *	12/2001	Warmus et al.	707/517

* cited by examiner

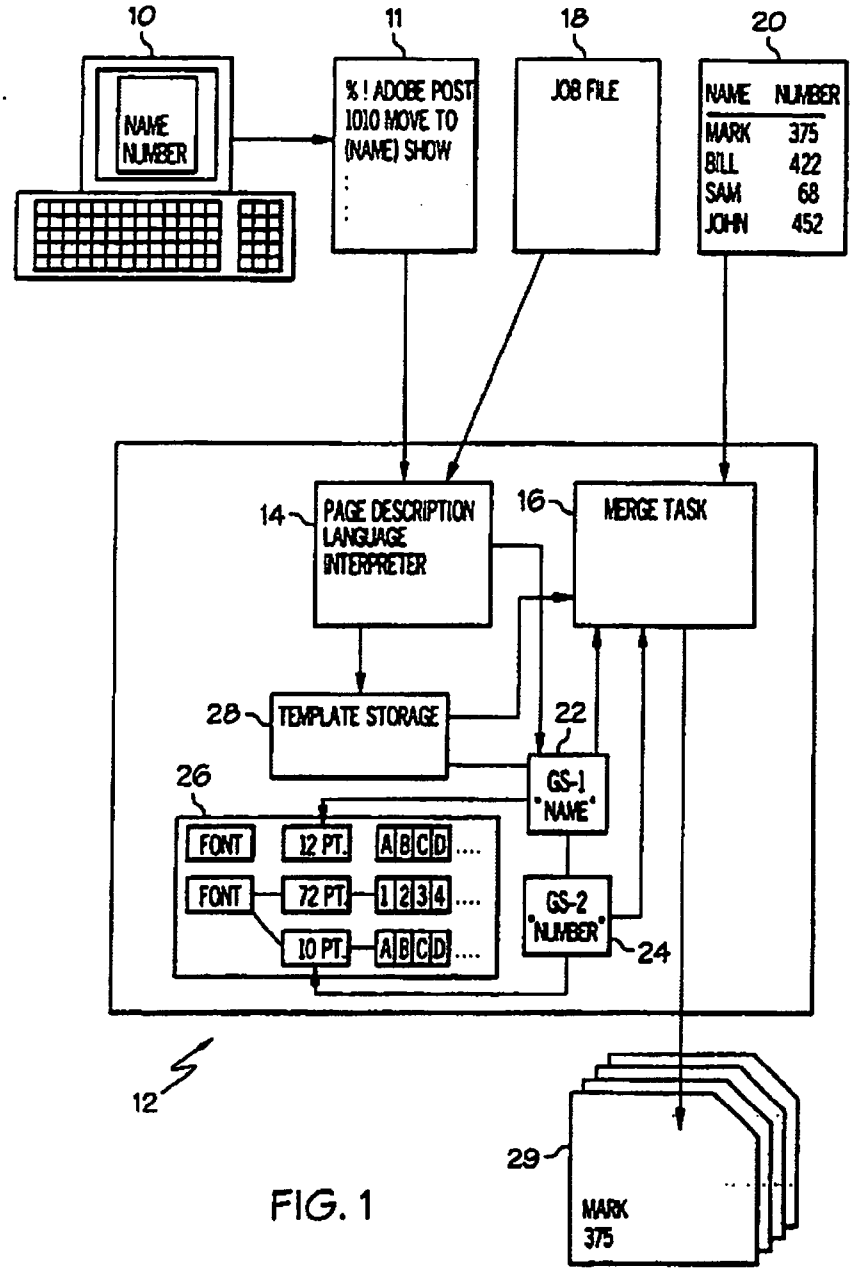

10
NAME
NUMBER
11
%! ADOBE POST
1010 MOVE TO
(NAME) SHOW
18
JOB FILE
20
NAME NUMBER
MARK 375
BILL 422
SAM 68
JOHN 452
14
PAGE DESCRIPTION
LANGUAGE
INTERPRETER
16
MERGE TASK
28
TEMPLATE STORAGE
22
GS-1
"NAME"
26
FONT
12 PT.
A B C D ....
FONT
72 PT.
1 2 3 4 ....
10 PT.
A B C D ....
GS-2
"NUMBER"
24
12
29
MARK
375

FIG. 1

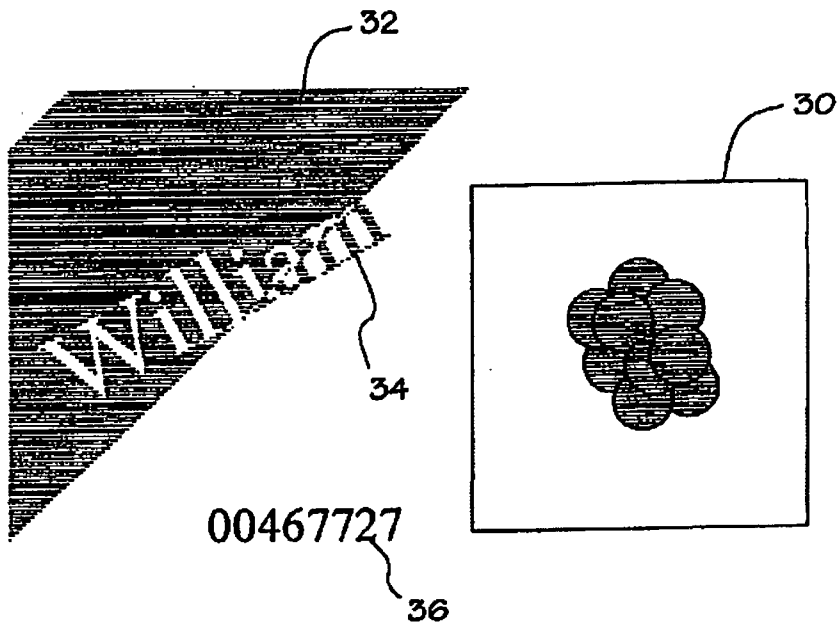


FIG. 2

METHOD OF UTILIZING VARIABLE DATA FIELDS WITH A PAGE DESCRIPTION LANGUAGE

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is a continuation of U.S. application Ser. No. 09/299,502, filed Apr. 26, 1999; which was a continuation of U.S. application Ser. No. 08/896,899, filed Jul. 18, 1997, and issued as U.S. Pat. No. 5,937,153; which is a continuation-in-part of U.S. application Ser. No. 08/373,582, filed Jan. 18, 1995, and issued as U.S. Pat. No. 5,729,665.

BACKGROUND OF THE INVENTION

The present invention relates to the high-speed printing industry, and more particularly, to a method for printing variable data using a page description language in a high-speed printing environment.

Application programs, such as Adobe Illustrator®, typically include a program which generates a specification of a screen or page's contents in a page description language. The specification, or page description code, provides instructions as to how to generate the image in a printer. The page description code is transferred from the application program to a printer, where it is executed to generate a bit map of the page. The most commonly used page description language is PostScript®, which is a machine independent language produced by Adobe Systems, Inc.

An application program page typically contains a number of data areas with either graphic or alphanumeric data. The PostScript language includes commands that define or build "graphics states" for each of the data areas on the page. These graphics states are sets of default attributes such as angle, scale factor, type-font, location, etc., which define how data is to appear on the page. Often, multiple graphics states are defined for a single page, with the different graphic states corresponding to different data areas on the page. Examples of commands that are used in PostScript to build a graphics state are: 20 rotate,/Times-Roman findfont, 14 scalefont, and setfont. In addition to commands which build graphics states, PostScript specifications also include the graphic or alphanumeric data which is displayed in the data areas, as well as a print command such as "SHOW", which causes a bit map to be generated for the data.

In the past, page description languages, including PostScript, have only been used to print static data pages, because page description languages lack the functionality required for variable data printing. In variable data printing, each page shares a common background, and the displayed data in at least one data field changes for each page. Up until now, it has not been possible to print pages of variable data with page description languages such as PostScript, because the page description languages are unable to save page backgrounds and graphics states from a page specification, and are thus unable reuse the same background and graphics states when printing subsequent pages. Thus, with page description languages such as PostScript, whether the entire page is changed, or only a single item of data on the page is changed, a new page description language specification is generated to print each separate page.

For example, if thousands of copies of a mass mailing advertisement were to be printed, each copy being identical except for the recipient's name and address, it would be necessary to generate a new PostScript specification defining the page background, and the graphics states for the name and address fields, for each new name and address that is printed. Hence, to print 50 advertisements, it would be necessary to generate 50 PostScript specifications which each define virtually the same image.

In general, PostScript specifications are very complex and require extensive processing to generate and execute. Thus, generating a new PostScript specification each time a page of variable data is printed consumes an immense amount of processing time. In high-speed printing systems, it is typically the processing time, not the printer hardware, which determines the speed at which pages can be printed. Therefore, the processing required to repetitively redefine the same background and graphics states for each page of variable data significantly slows the entire printing system.

Due to the amount of processing time consumed in redefining the page template and graphics states for each new page of data that is printed, as well as the resultant effect on printing speed, it is desirable to have a method for processing variable data wherein once defined, the template and graphics states for a page can be stored and reused for printing subsequent pages. Further, it is desirable to have a method for printing variable data which is compatible with existing printing systems and page description languages, such as PostScript, and which is capable of processing variable data in a high-speed industrial printing system.

SUMMARY OF THE INVENTION

It is an object of the present invention to provide a method for utilizing variable data with a page description language, which enables the template and graphics states for a page of variable data to be defined and stored; and which enables the stored graphics states to be associated with multiple items of variable data from a database or merge file; so that once stored, the graphics states can be repeatedly applied to the items of variable data to print multiple pages of variable data or multiple variable data bitmaps. Further, it is an object of the present invention to provide such a method which is compatible with existing page description languages, and which can be used in a high-speed industrial printing system.

The method of the present invention is implemented by means of a control task which executes in conjunction with a page description code interpretive program, such as a PostScript program, to identify variable data areas in the page description code specification, and reserve the graphics states for the variable data areas as they are defined by the specification. After the interpreter program has executed, a merge task is initiated. The merge task associates items of variable data from a data file with the reserved graphics states, generates a bit map for each variable data area, merges the bit maps with the page template, and outputs a complete bit map for the page. Accordingly, in the method of the present invention, bit maps for multiple pages of variable data are generated from a single page description language specification.

The present invention assumes the generation of a page specification in PostScript, or another similar page description language, by the application program, and the transfer of this specification to a printer. According to the present invention, a control task activates and monitors the PostScript interpreter program in the printer. As the interpreter executes, it defines graphics states for the data areas on the page. The PostScript attributes for a graphics state are stored in a stack as they are defined, so that at any given point in the code, the stack represents all of the PostScript attributes for the current graphics state.

When the control task identifies a print command in the code, the control task interrupts the interpreter to determine

whether the data to be printed is variable data. If the data is variable, the current graphics state, consisting of the attributes then existing in the stack and job specific attributes which are defined in a job file, is linked to the data area and reserved in an internal database. Further, character bit maps are generated in accordance with the graphics state, and linked to and reserved with the graphics state. After the graphics state and character bit maps have been reserved, the PostScript interpreter is resumed at the line of code following the print command.

The interpreter continues executing until either the control task detects another print command, or the last line of code is reached. If a second print command is detected, the interpreter is interrupted again and the above steps repeated, to reserve the stack contents and job attributes for the second data area, and to generate and store a second set of character bit maps. The control task continues in this manner monitoring and interrupting the interpreter program, until all of the variable data areas on the page have been detected, and graphics states and possibly character bit maps for the variable data areas have been reserved in the database.

As the PostScript interpreter executes, a bit map of the non-variable background graphics and text, otherwise referred to as a "template", is generated for the page. At the last code command, which in PostScript is typically "SHOWPAGE," the control task terminates the PostScript interpreter, and reserves the template in the database.

The merge task is then initiated to print variable data pages using the reserved page template, graphics states and character bit maps. The merge task begins by retrieving a merge file containing the variable data to be printed. After retrieving the merge file, the task identifies the correct template for the current page, and the names of the graphics states related to that template, from data in the merge file. Then, using the name of the first graphics state reserved for the template, the merge task retrieves the graphics state from the database and the character bit maps linked to that state. The merge task then retrieves data corresponding to that graphics state from the appropriate field in the merge file, and generates a bit map of the data in accordance with the graphics state and character bit maps. The merge task then merges the data bit map into the template. After the bit map has been generated and merged, the merge task identifies retrieves another graphics state for the template and repeats the process. If there are no more graphics states which correspond to variable data areas on the page, the merge task outputs the finished bit map for the page.

After the first page of data has been printed, the merge task retrieves a "clean" template from the database, and again identifies the graphics states for the page. The merge task then retrieves the next record of variable data from the database, and generates variable data bit maps for each of the fields in the record, in accordance with the reserved graphics states and character bit maps which correspond to each of the fields. The merge task continues in this manner, identifying variable data areas and generating bit maps for the variable data in the merge file, until a page has been printed for each variable data record in the file.

The method of the present invention is advantageous in that once the graphics states and template have been defined for a variable data page, they can be reused to print multiple pages of variable data with only a minimal amount of additional processing.

Accordingly, it is an object of the present invention to provide a method for printing variable data with a page description language; a method which increases the speed at which variable data pages can be printed; a method which enables the printing attributes for a page to be saved and used for printing multiple pages of data; and a method which is compatible with existing page description languages and printing systems.

Other objects and advantages of the present invention will be apparent from the following description, the accompanying drawings and the appended claims.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a data flow diagram illustrating the preferred embodiment of the method of the present invention;

FIG. **2** is an example of a variable data page generated in accordance with the method of the present invention.

DETAILED DESCRIPTION

The present invention provides a computer-implemented method for reserving graphics states, which enables a page description language to be used for variable data printing. In accordance with the present invention, graphics states describing variable data areas are generated by a page interpreter program and reserved in an internal database. The graphics states are later accessed from the database and used for printing variable data pages. The method of the present invention can be employed with a conventional page description language, such as PostScript, to enable variable data pages to be printed with a minimum amount of processing.

As shown in FIG. **1**, an image containing text and/or graphics data is created at a workstation **10**, using a graphics application program such as Adobe Illustrator®. As the image is created, the application program displays the image on the workstation screen. When the image is complete and ready to be printed as a page, the application program generates a specification of the image in PostScript in a conventional manner.

After the PostScript file **11** is generated, it is transferred from the workstation **10** to a printer generally designated as **12**. In the printer **12**, a PostScript interpreter **14** is executed to generate a pagemap of the image. In the method of the present invention, a control task operates in the printer **12** to initiate the PostScript interpreter program **14** and a merge task **16**. The control task is initiated upon power-on of the printer **12**, and controls and coordinates the variable data printing.

As the PostScript interpreter **14** executes, it defines the PostScript graphics state attributes for the page. These attributes can include the size, font, position, orientation, and location in which the graphic or text data is to appear on the page. The specifics of the attributes which are available in PostScript to define how data is to appear on a page would be well-known to one skilled in the art. Therefore, further description of these PostScript attributes is not provided.

However, one of the PostScript attributes, namely the region, has been expanded in the present invention to allow for variable data printing. In the method of the present invention, the region attribute is used to define the boundaries or extent to which a variable data graphics state will be allowed to effect a page. The graphics state extent is an invisible boundary which acts as a clippath for the page, to prevent variable data for a particular graphics state from extending outside the intended boundaries of the graphics state. The region of the graphics state extent is defined without altering PostScript, by using an ordinary shape, which is created as part of the image, to define the region.

In the present invention, the artist creates a shape representing the extent as part of the page image, and specifies a particular trigger color or gray scale for the shape. Thus, the artist could include a medium gray scale rectangle in the upper left-hand corner of the page, with the boundaries of the rectangle representing the extent which the artist has defined for the graphics state positioned at that corner of the page. The medium gray scale will then represent the trigger color, and will be specified as the trigger for the region attribute in a job file **18** in the printer **12**.

In addition, a second parameter in the job file **18** can be used to specify whether the rectangle should appear on the page, or whether it is being used only to define a graphics state extent. Thus, if the artist also wants the medium gray scale rectangle to appear on the printed page, this parameter enables the color to act as a trigger, yet not inhibit the artist's design. When the rectangle is interpreted during the method of this invention, the control task will detect the trigger color and will save an "invisible" boundary represented by the rectangular as part of the graphics state.

As the PostScript attributes are defined, they are placed in a stack. When a new attribute is defined, it is added to the top of the stack. When an attribute is deleted, it is removed from the stack. The combination of all of the attributes located in the stack at any point during the execution of the PostScript interpreter **14** constitutes the "current" graphics state for the page.

When the interpreter reaches a print command, such as "SHOW" in PostScript, the command triggers the control task to interrupt the interpreter program. During this interruption, the control task interprets data in the PostScript file **11** and reserves a graphics state if the data is variable. Normally in a PostScript file, data which is to appear on the printed document is enclosed within parentheses. Thus, the control task identifies data in the file **11** by locating parentheses in the code.

After the control task identifies the data, it interprets the data to determine whether it is static data, which is to be part of the page template, or variable data. To interpret the data, the control task first reads the data located in the parentheses and compares the data with a list of literal data strings stored in the job file **18**. The job file **18** contains a list of data strings which are each associated with the name of a graphics state and its corresponding data field in a merge file **20**. In the preferred embodiment, the graphics state name is the same as the field name in the merge file **20**. The merge file **20** contains variable data arranged in records, with each record corresponding to a different page. Each record contains one or more data fields, which each correspond to separate variable data areas on the page. The list of data strings and associated graphics state names is entered in the job file **18** by the print operator prior to initiating the print job. If the data from the PostScript file **11** matches a data string in the job file **18**, the control task replaces the data from the file **11** with the graphics state name associated with the matching data string. In this manner, the control task transforms static data in the PostScript file into a variable data field, by substituting a graphics state field name for the static data in the file.

In a second embodiment, the graphics state name corresponding to the data area is defined directly within the PostScript file **11**, by making the name part of the image that is created in the application program. In this embodiment, the name is enclosed within brackets in the file, such as "<<>>", to enable the control task to identify the data as defining a graphics state rather than being an ordinary data string. Thus, to define the graphics state "ADDRESS" within the PostScript file **11**, the following would appear before a show command in the code: "(<<ADDRESS>>)". This second embodiment is advantageous in that it does not require the control task to compare the file data with a data list in the job file **18**; however, it does require coordinating the graphics state and field names between the merge file **20** and the application program.

If the control task determines that the data corresponds to a variable area, it reads the current contents of the graphics state stack to determine the attributes to be used for printing data in that area. In addition to the PostScript attributes specified in the stack, the graphics state can also include attributes which are specifically tailored to variable data printing. These additional attributes can either appear after the graphics state name inside a "show" command, if the graphics state is defined directly in the PostScript file, or can be specified in the job file **18** prior to execution of the print job. These additional attributes specify how the variable data is to be positioned within the graphics state. The following is a list of the variable data attributes which can be specified for a print job:

Name:	A label used to identify the data to which the graphics state applies. A single datum may be inserted into more than one graphics state so this attribute is not unique to a single state.
Glyphs:	A list of character glyphs, both attributes and images, which are available for use in the graphics state. (e.g. an alphabet of 72 point Times-Roman bold italic characters).
Static Data:	Data to be used in the event that variable data is not available.
Identification:	A number used to uniquely identify a graphics state.
Justification:	How to handle the text left to right - left border, right border, centered or justified.
Alignment:	How to place the text vertically in the graphics state. This could be top, bottom or centered.
Word Wrapping:	Selects a word wrapping algorithm.
Dynamic Registration:	Information on how to determine the registration from one page to the next.
Logic Mode:	The manner in which the bitmap merge takes place. This is one of seven binary combination techniques.
DP Procedure:	A procedure (or program) used to manipulate the variable data just before the graphics state is applied.
Data Selection:	Which portions of the variable data to use.
Underline:	Selects underlined text.

When the control task is triggered to reserve a graphics state, the above listed attributes, if specified, are combined with the PostScript attributes from the stack, and reserved as a single graphics state under the name obtained from the PostScript file **11** or the job file **18** such as shown at **22**.

After the control task has compiled the attributes for the current graphics state, it may instruct PostScript to generate a font cache **26** for the graphics state. The font cache **26** consists of a character bit map for each of the alphanumeric

characters A–Z and 0–9 generated in the font specified in the graphics state. After PostScript has generated all of the character bit maps, and placed the bit maps in the font cache **26**, the font cache is linked to the graphics state **22**, and reserved in the database. After the control task has reserved the current graphics state **22** and the font cache **26** in the database, it resumes execution of the PostScript interpreter **14** at the first line of code after the print or "SHOW" command, so that the print command is not executed.

After the interpreter is resumed, it continues defining graphics state attributes for the page, until the control task detects another print or "SHOW" command. Upon detecting another print command, the control task again interrupts execution of the interpreter, and determines whether the data in the PostScript file **11** corresponds to a variable data area. If the data corresponds to a variable data area, the control task again substitutes a graphics state name from the job file **18** for the data in the PostScript file **11**, and reads the graphics state attributes from the stack and job file. The control task also instructs PostScript to generate another font cache, if the attributes of the current graphics state differ from the attributes of previously reserved graphics states. The current graphics state and font cache are then linked, and reserved in the database under the second graphics state name from the job file **18**, such as shown at **24**. If the data does not correspond to a variable data area, the control task resumes execution of the interpreter at the print command, so that a bit map for the data can be generated and added to the template.

At the final line of code, the template is complete, and incorporates all of the static text and graphic data that is to appear on the printed document. At this point, the control task terminates the interpreter, and saves the template to the database such as shown at **28**. In PostScript, the control task is triggered to save the template by the "SHOWPAGE" command.

Since the control task of the invention operates externally of the PostScript interpreter, the method of the present invention enables bit maps and graphics states to be generated by the interpreter in a conventional manner. However, rather than printing a completed page map at the end of the interpreter program, the method of this invention reserves the page maps, character bit maps and graphics states generated by the interpreter, in order that they may be subsequently accessed and used to print multiple pages of variable data.

After the interpreter has been terminated, the control task initiates the merge task **16**. The merge task **16** interfaces between the merge file **20**, which has been pre-programmed with items of variable data, and the database in which the templates, font caches and graphics states defined by the interpreter have been saved, in order to combine the variable data with a template on a single page map. The merge task **16** begins by accessing the merge file **20** to retrieve the name of the template for the page, and then retrieving the specified template from the database. In addition, the merge task **16** retrieves the names of the data fields and reserved graphics states which are associated with the selected template from the merge file **20**.

Using the name corresponding to the first graphics state on the page, the merge task **16** accesses the merge file **20** and retrieves the data stored under that field name in the first data record. In the representative merge file **20** shown in FIG. **1**, the field names are NAME and NUMBER.

After the merge task **16** has read the data corresponding to the designated field name, it retrieves the graphics state which was reserved under the same name, as well as the character bit maps which are linked to that graphics state. The merge task **16** then generates a bit map of the data in accordance with the graphics state attributes. After the bit map is generated, it is merged into the template at the region corresponding to the graphics state, by writing the data bit map over the existing template bit map.

It will be apparent to those of ordinary skill in the art that it is within the scope of the invention to write the data bit map over a clean page as opposed to the template bitmap. For example, if the template contains no static bitmap data, then it would not be necessary to save an empty bitmap of the template in the database as described above. Thus, it is within the scope of the invention that the PostScript file **11** defines only variable data areas and does not define any static data areas. Such a PostScript file is illustrated in FIG. **1**.

After the data from the first field has been merged into the template, the merge task **16** reads the name corresponding to a second variable data area from the merge file **20**, if a second variable area exists on the page. The merge task **16** then retrieves the graphics state and linked font cache having the same name as the second variable area. Next, using this name, the merge task **16** again accesses the merge file **20**, and reads the data from the field of the same name. The merge task **16** then generates a bit map for the data in accordance with the graphics state and font cache, and again merges the data bit map into the template **28**.

The merge task **16** continues the steps of identifying variable data areas for the template, retrieving graphics states and character bit maps corresponding to the variable areas, accessing variable data from the merge file **20**, and generating bit maps for the variable data, until bit maps have been generated and merged for all of the variable data to be included on the page. When a bit map has been generated for each variable data area, and merged with the template **28**, the pagemap is output for printing as shown at **29**.

The merge task **16** then proceeds with printing a second page using the same template and graphics states, but a different variable data record in the merge file **20**. To print the second page, the merge task **16** retrieves a "clean" template from the database. Next, the merge task **16** again identifies the name of the first variable data area for that template and retrieves the graphics state of the same name. Then, the merge task **16** reads the data for that field from the second record of the merge file **20**, and generates a bit map of the data using the retrieved graphics state attributes and character bit maps. Once the bit map is generated, the merge task **16** merges the bit map into the template by writing the bit map over the template at the location defined by the graphics state.

The merge task **16** then continues processing in this manner until bit maps have been generated and merged into the template for all of the graphics states reserved for the page. After all of the bit maps for the second page have been merged into the template, the page is printed. The merge task **16** continues, repeating these steps for each record of data in the merge file **20**, until all of the variable data records have been printed on a page.

FIG. **2** shows a variable data page printed in accordance with the method of this invention. On this page, the data fields **30** and **32** are static fields which are part of the page template. The data field **34** containing the name "William" is a variable data field. Different names such as Mark or Sam, from the merge file **20**, are printed in this field on subsequent pages. The font, angle and color contrast in

which "William" is displayed are all aspects of the graphics state which were defined and stored during the steps of the present invention. Data field **36** which contains the number "00467727" is a second variable data area on the page. Again, the data displayed in this area varies on each page, depending upon the contents of the merge file **20**.

While the method described constitutes a preferred embodiment of the invention, it is to be understood that the present invention is not limited to this precise form, and that variations may be made without departing from the scope of the invention.

What is claimed is:

1. A computer implemented method for generating a plurality of bitmaps suitable for printing comprising the steps of:

(a) interpreting a page description code specification, the page description code specification defining at least one template bitmap and including at least one data area into which at least one of a plurality of a variable bitmaps is to be merged, the interpreting step including a step of identifying the data area;

(b) producing a first template bitmap;

(c) merging at least a first one of the plurality of variable bitmaps into the one data area of the first template bitmap to create a first merged bitmap;

(d) printing the first merged bitmap;

(e) producing a next template bitmap;

(f) merging at least a next one of the plurality of variable bitmaps into the one data area of the next template bitmap;

(g) printing the next merged bitmap; and

(h) repeating steps (e) through (g) for the plurality of variable bitmaps.

2. The computer implemented method of claim **1**, wherein the plurality of variable bitmaps are a plurality of character bitmaps.

3. The computer implemented method of claim **2**, wherein the interpreting step includes the step of creating the plurality of character bitmaps.

4. The computer implemented method of claim **3**, wherein the step of creating the plurality of character bitmaps includes the step of applying an attribute in the page description code specification associated with the data area to a plurality of characters.

5. The computer implemented method of claim **4**, wherein the attribute defines an aspect of how data is to appear in the data area.

6. The computer implemented method of claim **5**, wherein the attribute is a graphics state defined for the data area.

7. The computer implemented method of claim **5**, wherein the attribute defines a position of the variable data bitmap with respect to the template bitmap.

8. The computer implemented method of claim **5**, wherein the attribute defines an orientation of the variable data bitmap with respect to the template bitmap.

9. The computer implemented method of claim **1**, wherein the interpreting step includes the step of creating the plurality of variable bitmaps.

10. The computer implemented method of claim **9**, wherein the step of creating the plurality of variable bitmaps includes the step of applying an attribute in the page description code specification associated with the data area to a plurality of variable data items.

11. The computer implemented method of claim **10**, wherein the attribute defines an aspect of how bitmap data is to appear in the data area.

12. The computer implemented method of claim **11**, wherein the attribute is a graphics state defined for the data area.

13. The computer implemented method of claim **11**, wherein the attribute defines a position of the variable data bitmap with respect to the template bitmap.

14. The computer implemented method of claim **11**, wherein the attribute defines an orientation of the variable data bitmap with respect to the template bitmap.

15. The computer implemented method of claim **10**, wherein the step of creating the plurality of variable data bitmaps further includes the step of applying an attribute defined in an external file to the plurality of data items.

16. The computer implemented method of claim **15**, wherein the step of creating the plurality of variable data bitmaps further includes the step of associating the attribute defined in the external file with the data area.

17. The computer implemented method of claim **1**, wherein the step of producing the first template bitmap includes the step of processing a portion of the page description code specification defining the at least one template bitmap to generate the first template bitmap.

18. The computer implemented method of claim **17**, wherein the step of producing the first template bitmap further includes the step of storing the first template bitmap in memory.

19. The computer implemented method of claim **18**, wherein the step of producing the next template bitmap includes the step of accessing a copy of the first template bitmap stored in memory.

20. The computer implemented method of claim **17**, wherein the step of processing the a portion of the page description code specification defining at least one template bitmap to generate the first template bitmap further includes the step of not processing a portion of the page description code specification defining the at least one data area into which at least one of a plurality of a variable bitmaps is to be merged.

21. A computer implemented method for generating a plurality of bitmaps suitable for printing comprising the steps of:

providing a page description code specification defining at least one template bitmap and including at least one data area into which at least one of a plurality of bitmaps is to be merged;

providing a file external to the page description code specification including at least one attribute item defining an aspect of how bitmap data is to appear on a printed page;

processing the page description code specification to generate a template bitmap, and during the processing step, identifying the data area;

upon identifying the data area, applying the attribute defined by the attribute item in the external file to a first one of the plurality of bitmaps; and

merging the first one of the plurality of bitmaps with the template bitmap.

22. The computer implemented method of claim **21**, further comprising the step of, prior to applying the attribute defined by the attribute item in the external file, associating the data area with the attribute item in the external file.

23. The computer implemented method of claim **21**, wherein the attribute defined by the attribute item in the external file is horizontal justification attribute.

24. The computer implemented method of claim **21**, wherein the attribute defined by the attribute item in the external file is a vertical alignment attribute.

25. The computer implemented method of claim 21, wherein the attribute defined by the attribute item in the external file is a word wrapping algorithm.

26. The computer implemented method of claim 21, wherein the attribute defined by the attribute item in the external file is a binary logic designation defining how the first one of the plurality of bitmaps is to be merged with the template bitmap.

27. The computer implemented method of claim 21, wherein the attribute defined by the attribute item in the external file is an underlined text attribute.

28. A computer implemented method for generating a plurality of bitmaps suitable for printing comprising the steps of:

providing a page description code specification defining at least one template bitmap and including at least one data area into which at least one of a plurality of variable data bitmaps is to be merged;

providing a job file external to the page description code specification including at least one attribute item defining an aspect of how bitmap data is to appear on a printed page;

providing a merge file external to the page description code specification include a plurality of variable data items;

processing the page description code specification to generate a template bitmap, and during the processing step, identifying the data area;

upon identifying the data area, applying the attribute defined by the attribute item in the job file to a first one of the variable data items in the merge file in the generation of a variable data bitmap; and

merging the variable data bitmap with the template bitmap.

29. The computer implemented method of claim 28, wherein the attribute defined by the attribute item in the job file is horizontal justification attribute.

30. The computer implemented method of claim 28, wherein the attribute defined by the attribute item in the job file is a vertical alignment attribute.

31. The computer implemented method of claim 28, wherein the attribute defined by the attribute item in the job file is a word wrapping algorithm.

32. The computer implemented method of claim 28, wherein the attribute defined by the attribute item in the job file is a binary logic designation defining how the variable data bitmap is to be merged with the template bitmap.

33. The computer implemented method of claim 28, wherein the attribute defined by the attribute item in the job file is an underlined text attribute.

34. The computer implemented method of claim 28, wherein the attribute item in the job file defines a procedure to manipulate the variable data item.

35. The computer implemented method of claim 28, wherein the attribute item in the job file defines a static data item to be used in the generation of the variable data bitmap if the variable data item is not available.

36. A computer implemented method for generating bit maps suitable for printing comprising the steps of:

(a) providing a page description code specification, the page description code specification defining at least one data area;

(b) identifying the data area defined by the page description code specification;

(c) upon identification of the data area in step (b), associating with the data area an attribute defining an aspect of the appearance of data in the data area, and applying the attribute to a set of characters so as to generate a plurality of character bitmaps;

(d) associating a first variable data item from a set of variable data items with the character bitmaps; and

(e) generating a first variable data bitmap for the first variable data item using the character bitmaps.

37. The computer implemented method of claim 36, further comprising the steps of:

(f) associating a next variable item from the set of variable data items with the character bitmaps;

(g) generating a next variable data bitmap for the next variable data item using the character bitmaps; and

(h) repeating steps (f) and (g) for a plurality of the variable data items in the set of variable data items.

38. The computer implemented method of claim 37, wherein the page description code specification represents a template and includes a static data area, and the computer implemented method further comprises the steps of:

executing portions of the page description code specification corresponding to the static data area to generate a template bitmap;

merging the first variable data bitmap into the template bitmap or a copy of the template bitmap to generate a first merged bitmap; and

merging the next variable data bitmaps into the template bitmap or a copy of the template bitmap to generate a plurality of next merged bitmaps.

39. The computer implemented method of claim 36, wherein the page description code specification represents a template and includes a static data area, and the computer implemented method further comprises the steps of:

executing portions of the page description code specification corresponding to the static data area to generate a template bitmap; and

merging the first variable data bitmap into the template bitmap or a copy of the template bitmap to generate a first merged bitmap.

40. The computer implemented method of claim 36, wherein the attribute defining an aspect of the appearance of data in the data area is provided in the page description code specification.

41. The computer implemented method of claim 40, wherein the attribute defining an aspect of the appearance of data in the data area is an attribute associated with the data area.

42. The computer implemented method of claim 36, wherein the attribute defining an aspect of the appearance of data in the data area is provided in a file external to the page description code specification.

* * * * *